ELSEVIER

Discrete Applied Mathematics 63 (1995) 75–89

DISCRETE APPLIED MATHEMATICS

Drawing graphs on rectangular grids

Markus Schäffter*

Technische Universität Berlin, Strasse des 17 Juni 135, D-14623, Berlin, Germany

Received 2 March 1993; revised 10 December 1993

Abstract

We consider the problem of embedding undirected graphs $G = (V, E)$ with n vertices of maximum degree 4 (4-*graphs*) into rectangular grids. The problems of minimizing the grid area (Kramer and van Leeuwen, 1984; Formann and Wagner, 1991) as well as minimizing the total number of bends are NP-hard (Storer, 1984). It is well known that such a graph can be embedded in a rectangular grid with at most $2n$ columns and $2n$ rows such that any edge is embedded with at most 5 bends (Lengauer, 1990, p. 249).

We present an $\mathcal{O}(n^2)$-algorithm constructing an embedding for arbitrary 4-graphs with n vertices in a rectangular grid with at most $2n$ columns and $2n$ rows bending every edge at most twice. Note that two is a lower bound on the maximum number of bends per embedded edge since the graph K_5 cannot be embedded with less than 2 bends per embedded edge.

1. Introduction

Building layouts of integrated circuits on chips is quite difficult. The larger the area covered by a circuit, the more difficult is the production of trouble-free chips. Thus, minimizing the area covered by a circuit is a natural effort in VLSI-Design. In practice, other parameters play a role as well. The lengths of the wire (total sum or maximum length), the number of crossings between wires and the number of bends per wire. However, area always tends to be the most important parameter. The state-of-the-art approach is to route wires along the edges of a grid in order to avoid induction between parallel edges.

A very simple layout problem is to find an edge-disjoint embedding of a circuit that can be represented by a graph (with a maximum vertex degree of four) in a rectangular grid. But finding an embedding of minimum area or minimizing the total number of bends are NP-hard problems even in this simple case (see [6, 3, 10]).

Thus it is an interesting question whether there exist upper bounds for the grid area. For the case of planar graphs, Kant presents the following results [5, 4].

* E-mail: shefta@math.tu-berlin.de.

SSDI 0166-218X(94)00020-E

- Every triconnected planar graph with degree at most four can be drawn orthogonally on a grid of size at most $n \times n$ with at most $\lceil 3n/2 \rceil + 4$ bends such that every edge has at most 2 bends if $n > 6$.
- Every planar graph with degree at most three can be drawn orthogonally on a grid of size at most $\lfloor n/2 \rfloor \times \lfloor n/2 \rfloor$ with at most $\lfloor n/2 \rfloor + 1$ bends if $n > 4$.

Further results for planar graph embedding (especially non-orthogonal embedding) can be found in [5]. This dissertation also gives a very fine comprehensive overview on recent results in the field of planar embedding. For arbitrary graphs, few results are known. Lipton and Tarja have shown that there exist graphs in size $\mathcal{O}(n)$ that require $\Omega(n^2)$ crossings and hence as many vertices in any planar representation [11, 9].

The first part of this paper presents an algorithm that constructs embeddings with at most three bends per wire into a $2n \times 2n$ grid. Moreover, the resulting embeddings can be modified to obtain at most two bends per wire without enlarging the grid area. The necessary modification is treated in the second part. Applying the algorithm to the graph K_5, we show how the single steps of the algorithm work in detail.

2. An embedding with at most three bends per wire

The main idea of the algorithm is to embed the vertices $v_1, \ldots, v_n$ of the given graph G diagonally in a $2n \times 2n$ grid, beginning at the upper left corner, reserving for each vertex v_i a unit square V_i (see Fig. 1). This idea was described by Valiant [11], but he focused on planar graphs. In a first step, we embed the edges of G as paths along grid edges between grid nodes on the borders of these unit squares. We call this first step the *routing of the connections*. Then, in a second step, we extend these paths along the squares' borders such that all paths incident to a square V_i meet in one common grid node on the border of V_i, the *embedding point of the vertex* v_i. There is a one-to-one correspondence between vertices of G and their embedding points in the grid. Because of that, we denote both by v_i.

Two embedding points will be connected in the grid if and only if there exists an edge in the graph between the corresponding vertices. Because of that, we call a path

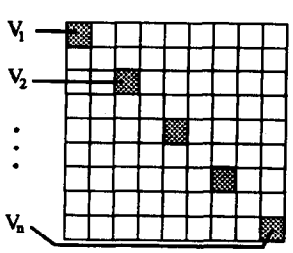

Fig. 1. ($n = 5$).

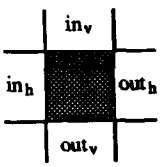

Fig. 2. The eight contacts of a square in the grid.

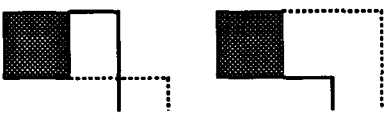

Fig. 3. How two neighbouring connections can exchange their contacts.

in the grid between two embedding points v_i and v_j the *embedding of the edge* (v_i, v_j) *in the grid* or the *wire* (v_i, v_j) for short. We denote a path between the borders of two squares V_i and V_j, with $i < j$, by (V_i, V_j) and call it *the connection of* V_i *and* V_j. For a connection between the squares V_i and V_j, with $i < j$, we call V_i the *start square* and V_j the *target square* of this connection.

Note that all unit squares are incident to different grid edges. Thus, we call the eight grid edges incident to a unit square V_i the *contacts* of V_i. All connections between squares have to be routed over these contacts. We will realize the connections between the squares' borders step-by-step, beginning with V_1, proceeding up to V_{n-1}, routing in the kth step all connections between V_k and its neighbours in $\{V_{k+1}, \ldots, V_n\}$. Thereby, we always start at the right or at the lower border of V_k and finish at the left or at the upper border of the corresponding target square. Thus, we call the grid edges incident to the left or to the upper border of a square V_i the *input contacts of* V_i or the *inputs of* V_i for short. Similarly, we call the grid edges incident to the right or to the lower border of a square V_i the *output contacts of* V_i or the *outputs of* V_i for short. We distinguish *horizontal* and *vertical inputs/outputs* and denote them as shown in Fig. 2.

Notice that the square V_1 will only use output contacts and that the square V_n will only use input contacts. Realizing a connection, we always start at a horizontal output and finish at a vertical input or start at a vertical output and finish at a horizontal input, respectively. Hence, all resulting connections can be routed in the grid with exactly one bend outside the squares.

Since all squares will use different rows and columns of the grid for their connections, it makes no difference whether we use the upper or the lower horizontal output of a square. As we will see in the next section, a good choice of the covered contacts allows an embedding with at most two bends per wire. Thus, it can be useful to exchange two outputs as shown in Fig. 3.

To be able to find a proper embedding point, we have to route the connections of each square carefully. Fig. 4 gives a situation which does not allow to extend the

Fig. 4. Connections that cannot be extended properly.

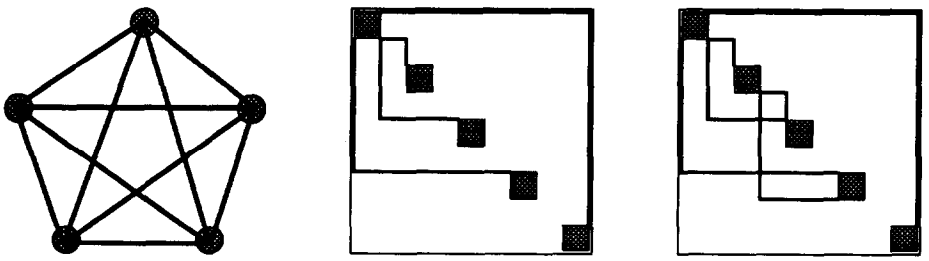

Fig. 5. The graph K_5 and the first two steps of the algorithm.

connections in an edge-disjoint way to one common grid node on the square's border.

To prevent such situations, we *balance* the ingoing and the outgoing connections of each square. We call a contact *covered* if it is used by connection and *non-covered* or *free* otherwise. A square is called *balanced with respect to its inputs* if $|\#\{\text{covered } in_h\} - \#\{\text{covered } in_v\}| \leqslant 1$. It is called *balanced with respect to its outputs* if $|\#\{\text{covered } out_h\} - \#\{\text{covered } out_v\}| \leqslant 1$ and just *balanced* if it fulfills both conditions.

We will realize the connections in such a way that all squares are kept balanced in every step. Consider the graph K_5 as an example. Fig. 5 shows the first two steps of routing connections between the squares. In the first step, all connections of V_1 are routed, keeping all squares balanced. In the second step, the remaining 3 connections of the square V_2 are routed. The picture on the right side of Fig. 5 gives a possible routing of two outgoing connections of V_2. Note that in this situation the routing of the third connection leaving V_2 violates the balance condition of the square V_4. Thus, before routing the last connection of V_2, the balance of V_4 has to be reestablished. Therefore it is necessary to reroute connections. Fig. 6 gives an example of such a rehang modification, and Fig. 7 shows how the balance of the square V_4 is reestablished in our example.

Note that such a modification can always be done since different squares use different rows and columns in the grid. The following lemma shows how such a modification can be realized.

Lemma. *Consider a square S with exactly two ingoing connections, both at the upper border of S. Call one of the covered vertical inputs* c_v *and one of the uncovered horizontal*

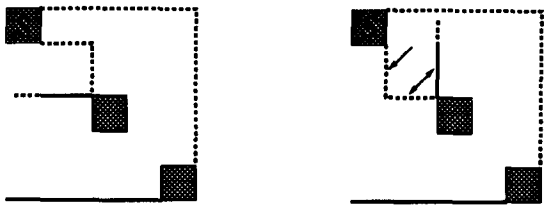

Fig. 6. Reestablishing the balance of the square S.

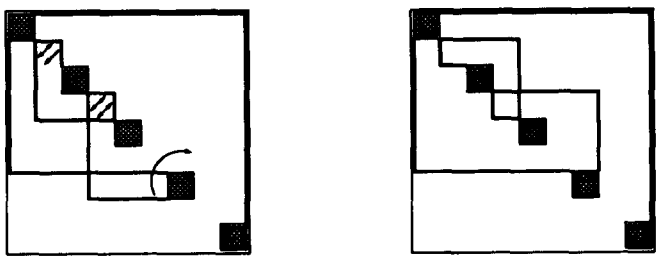

Fig. 7. Reestablishing the violated balance of V_4.

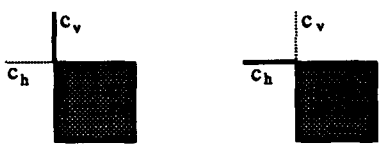

Fig. 8. A possible exchange of two contacts.

inputs c_h. Then, under the condition that all the other squares are balanced, there exists another realization of the already routed connections that covers the same inputs/outputs on all squares with two exceptions. On the square S, the contact c_h is covered and c_v is not.

Remark. The above Lemma shows that, if all squares with the exception of S are balanced, it is possible to exchange two inputs of S, such that S fulfills the balance condition as well (see Fig. 8). We call such a modification a *rehanging of two inputs.* Note that because of symmetry, the above lemma remains true in the case of 2 uncovered vertical and 2 covered horizontal inputs.

Proof. As mentioned above, we want to "rehang" the connection covering the input c_v such that it covers c_h instead. Let (S', S) be the connection covering c_v. It leaves the

square S' on a horizontal output since it covers a vertical input of S. Now, we move (S', S) from its horizontal output to a vertical output. To maintain the balance of S', it can be necessary to move another connection from its vertical output (which then can be used for (S', S)) to the horizontal output of (S', S). Iterating this process of exchange will give another embedding where (S', S) covers the input c_h. The fact that this iteration stops follows from the fact that all squares but S are balanced. □

To simplify the analysis, we just store the exchanges of a rehang phase, realizing all exchanges at the end of the phase at once. The following algorithm executes such a rehang phase.

Algorithm Router

//c_h is a non-covered input and c_v is a covered input. The case "c_v non-covered and c_h covered" can be solved similarly.//

(1) Store the exchange of the inputs c_h and c_v in the square S and mark c_h and c_v as *rehanged.*

(2) Follow the connection (S', S) covering the vertical input c_v. This leads us to a horizontal output c'_h of S'. Call S' the *current square.*

(3) Find in the current square S' a covered and unmarked (this means not yet rehanged) vertical connection c'_v corresponding to c'_h. "Corresponding" means both c'_h and c'_v have to be inputs or both have to be outputs. If there exists no such covered connection, there must be a corresponding non-covered connection c'_v.

(4) Store the exchange of c'_h and c'_v in S' and mark them to be *rehanged.*

(5) If c'_v is non-covered, realize all stored exchanges and STOP.
Otherwise, follow the connection (S'', S') covering the vertical contact c'_v to its horizontal contact c''_h on a square S''. Let S'' be the current square, denote S'' by S' and c''_h by c'_h and proceed with step (3).

An invariant of Router

If we follow a connection in step (2) *or step* (5), *we reach squares only via horizontal connections* c'_h *that have not previously been examined* (*marked*).

Since the number of connections of a square is even and because of the construction of step (3), we will always find a corresponding connection c'_v in the current square. If we realize all stored exchanges, the corresponding connections are still routable (this means they start at horizontal (vertical) output contacts and finish at vertical (horizontal) input contacts) because we exchange only horizontal inputs (outputs) with vertical inputs (outputs) (see Fig. 9).

The invariant also guarantees the termination of the algorithm Router. Since each square has at most eight connections, we pass each square at most four times. Note that an execution of Router exchanges only covered connections (except the first and the last exchange) and that all squares (except S') remain balanced since step (3) prefers covered connections c'_v over free ones.

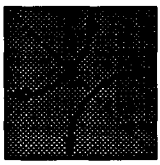

Fig. 9. Possible exchanges of connections done by Router.

Using algorithm Router, we are able to realize all the connections between the borders of the squares $V_1, \ldots, V_n$ step-by-step. In the kth step we route the remaining connections of the square V_k via its outputs. We do this in such a way that V_k remains balanced with respect to its outputs. Proceeding in the order $V_1, \ldots, V_n$, we have to route in the kth step only connections from V_k to target squares in $\{V_{k+1}, \ldots, V_n\}$. If the routing of a connection (V_k, V_l) violates the balance of V_l (with respect to the inputs), we use Router to reestablish the balance condition on V_l. We call this algorithm the Balancer algorithm.

Algorithm Balancer

- Proceed in the order $V_1, \ldots, V_n$ and route in the kth step the remaining connections of V_k via the outputs of V_k. If there remain exactly two connections, realize one via a horizontal and the other one via a vertical output to maintain the balance of V_k. If the routing of a connection (V_k, V_l) destroys the balance of the square V_l, reestablish the balance of V_l using the algorithm Router.
- Extend all connections along the squares' borders.

An invariant of Balancer

In the kth step, all edges in the graph G incident to the vertices $v_1, \ldots, v_{k-1}$ are already embedded in the grid (between the squares' borders) such that all squares are balanced.

This invariant shows that, for arbitrary graphs with a maximum vertex degree of 4, Balancer computes an embedding that fulfills the balance condition on all squares $V_1, \ldots, V_n$. We call such an embedding *balanced*. The Balancer algorithm first routes all connections between the squares $V_1, \ldots, V_n$ with respect to the balance condition. Then, in a second step, Balancer extends the routed connections along the squares' borders in such a way that all connections of a square V_i meet in one common grid node on the border of V_i, the embedding point of the vertex v_i. With respect to symmetry and with respect to the balance condition, there are four different cases that can occur while extending the connections along the border of a square.

(a) A square with 4 covered inputs and with no covered outputs.
(b) A square with 3 covered inputs and with 1 covered output, the output is opposite to exactly one input.
(c) A square with 3 covered inputs and with 1 covered output, the output is opposite to exactly two inputs.
(d) A square with 2 covered inputs and with 2 covered outputs.

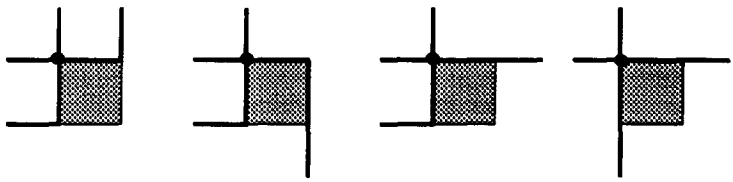

Fig. 10. Extensions along a square's border.

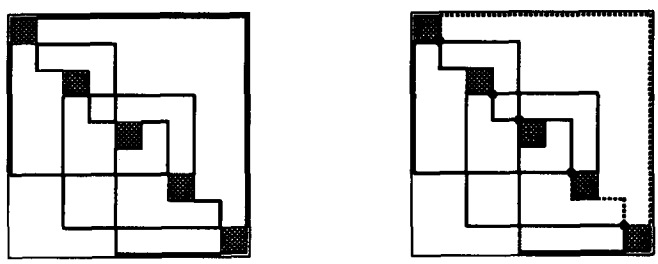

Fig. 11. A balanced routing and an embedding for the K_5.

Note that case (a) is similar to the case "no covered inputs/4 covered outputs" with respect to symmetry and that there are two cases for "1 input/3 outputs" similar to (b) and (c). Fig. 10 shows extensions of the connections to a common embedding point for the all above cases.

Extending the connections as shown in Fig. 10, each wire will have at most three bends, one on the border of the start square, one on the border of the target square and one bend between those squares. Thus, the algorithm Balancer computes an embedding with at most three bends per wire. But we can do better. Preventing a wire having bends both on its start as well as on its target square will improve this bound to at most two bends per wire. This is optimal for *arbitrary* graphs since the graph K_5 cannot be embedded in a grid without allowing at least two bends per wire (see Fig. 11).

3. An embedding with at most two bends per wire

As we have seen in the above section, the routing produced by the first step of the Balancer algorithm can be extended along the squares' borders in such a way that each wire has at most three bends. As mentioned before, preventing a wire having bends both on its start square, as well as on its target square, will improve this bound to at most two bends per wire. The main idea to realize this is as follows. We present an algorithm that computes a colouring of the wires that induces an embedding with at most 2 bends per wire. We call such a colouring *feasible*. The algorithm works

♯ covered inputs	0	1	2	3	4
♯ covered outputs	4	3	2	1	0
♯ green outgoing connections	2	2	2	0	0

Fig. 12. Minimal number of green outgoing connections.

Fig. 13. Extending critical inputs.

step-by-step, beginning with the square V_1, proceeding to V_{n-1} (note that V_n has no outgoing connections), colouring in the kth step all the connections leaving V_k. Thereby, we mark outgoing connections that can be routed without a bend *green* and the others *red*. Thus, the ingoing connections of a square V_k that are marked *green* may have a bend on the border of V_k, while *red* ingoing connections have to be extended along the border of V_k without a bend. For this reason, we call *green outgoing connections* and *red ingoing connections* of a square S *critical connections* since they have to be routed without a bend along the border of S. In the following figures, we denote red connections by dashed lines while bold solid lines denote green connections.

For all the extensions given in Fig. 10, at most two of the inputs can be extended without a bend on the square's border. Thus, each square may have at most two red inputs and those inputs have to lie on different borders of the square. The number and the positions of outgoing connections that can be coloured green, i.e. that can be extended without a bend on the square's border, depends on the number and on the positions of the red coloured ingoing connections. Fig. 12 shows how many outgoing connections can be coloured green at least.

In every case, the critical ingoing (outgoing) connections of each square have to be *balanced* in such a way that two critical inputs (outputs) do not lie on the same border of a square. Fig. 13 shows two critical ingoing connections that can (left side) and two that cannot (right side) be extended to a common grid node without a bend.

Let us proceed with our example. Fig. 11 shows a balanced routing of all connections, while Fig. 14 presents a feasible colouring of this routing and the resulting embedding of the graph K_5 with at most two bends per wire. Since *green outgoing* connections have to be routed without a bend, it is necessary to exchange the two connections on the right vertical border of the square V_1 on the upper left corner of the embedding space.

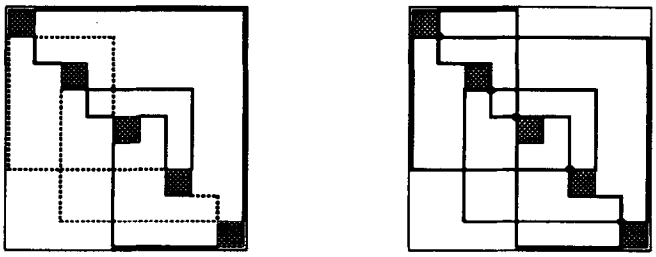

Fig. 14. A feasible colouring and an embedding with at most two bends per wire.

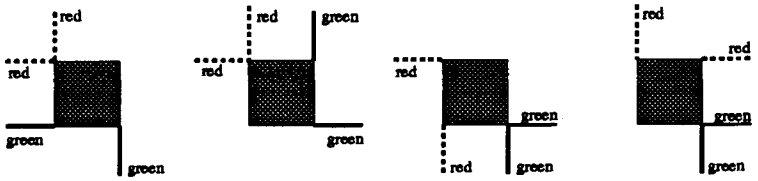

Fig. 15. Forbidden situations.

In order to construct a routing with at most two bends per wire, we will first find a colouring fulfilling the following conditions. A colouring like this will then show us the right way how to embed the given graph.

Colouring conditions

(1) All squares have at most two red ingoing and at most two green outgoing connections.
(2) Two red ingoing (green outgoing) connections never lie on the same side of a square. (In this sense, the critical connections are balanced.)
(3) All squares remain balanced with respect to their inputs/outputs.
(4) The situations in Fig. 15 never occur.

The following lemma guarantees the existence of a feasible colouring under the condition that all squares are balanced with respect to their inputs/outputs.

Colouring lemma. *Consider a routing of all connections such that all squares are balanced with respect to their inputs/outputs. Then a connection that is not already marked can be coloured in such a way that the colouring conditions* (1)–(4) *are fulfilled for all squares* $V_1, \ldots, V_n$.

Proof. The following algorithm Painter marks the connections step-by-step, colouring in the kth step the outgoing connections of V_k in such a way that the colouring conditions (1)–(4) are fulfilled. There is only one problem to consider. If an outgoing connection c_r is marked red, one of the situations mentioned in Fig. 16 can occur on

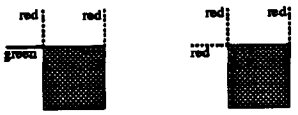

Fig. 16. Two red inputs at the same border violate condition (2).

the target square of the corresponding connection, violating colouring condition (2). To handle these situations, we present an algorithm, called Chameleon, which re-colours already marked connections if necessary.

Algorithm Chameleon

Input: A square S with an ingoing connection c_r (unmarked or green) that shall be coloured red.

(1) Colour the ingoing connection c_r red.

(2) If there is a red ingoing connection $c'_r = (S', S)$ at the side of c_r on the square S, colour c'_r green and let S' be the current square. Apply the following two rules.

Rule 1: There is a green outgoing connection $c'_g = (S', S'')$ beside c'_r on S' violating colouring condition (2).

Colour c'_g red and if this violates the colouring condition (2) on the square S'' use Chameleon again relative to S'' and the connection c'_g.

Rule 2: Colouring c'_r green violates colouring condition (4).

In this case, c'_r is an outgoing connection of S' which lies opposite to a red coloured ingoing connection c''_r. Mark c''_r *green* and let the start square of c''_r be the current square. Continue applying the above rules relative to the current square until Rule 1 can be applied.

The termination of the algorithm Chameleon follows from the fact that each square is passed at most twice. To see this, consider the way the algorithm passes the squares. In Rule 1, the algorithm changes the colour of two connections on the same border of Square S'. In Rule 2, it passes Square S' diagonally between two borders with exactly one connection. In both cases, coming from a connection that was handled the first time, the algorithm Chameleon switches to another connection that was never handled before.

Note that the connections, Chameleon switches to, either are *incoming red* or *outgoing green* ones. Since the first connection c_r is coloured *red* at the very beginning, there is no way, that a connection already handled by Chameleon will be passed a second time. Since all squares have at most four connections, each square can be passed at most twice.

Now, we are able to formulate an algorithm that realizes the colouring of all connections.

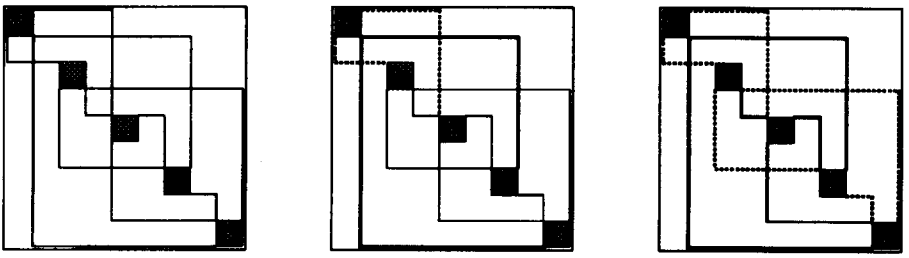

Fig. 17. A routing for the graph K_5 produced by Balancer and the first two steps of Painter.

Algorithm Painter

Proceed in the order $V_1, \ldots, V_{n-1}$. In the kth step colour the outgoing connections of V_k. First, colour as many of the outgoing connections green as given in the table in Fig. 12. Then colour the remaining outgoing connections red such that the following two conditions are fulfilled.

(a) In the case of 1 covered input and 3 covered outputs, a green outgoing connection must not lie opposite to a red coloured ingoing connection (this establishes colouring condition (4)).

(b) Two green outputs must not lie on the same side of a square (this implies colouring condition (2)).

After colouring an outgoing connection (V_k, S) red, check whether the colouring conditions (1)–(4) are fulfilled for Square S. If one of the conditions is violated, use Chameleon w.r.t. Square S to restore it.

Invariants of Painter

In each step, the colouring conditions (1)–(4) *are fulfilled.*

Thus, when Painter terminates, each square has at most two red ingoing and at most two green outgoing connections (condition (1)) and red ingoing as well as green outgoing connections are balanced in the sense of condition (2). □

Let us consider the example given in Fig. 17. The picture on the left side shows a balanced routing of the connections for the graph K_5. In a first step, algorithm Painter colours all connections of V_1 (picture in the middle). The picture on the right side shows a conflict that occurs while colouring the outgoing connections of V_4 *red* (colouring condition (2) is violated).

The picture on the right side in Fig. 17 shows a situation in which colouring condition (2) is violated on the square V_5 (two red connections are entering). Starting the algorithm Chameleon colours first connection (V_4, V_5) *red*. To restore colouring condition (2), Chameleon then changes the colour of (V_2, V_5) from red to *green*. After that, colouring condition (4) is violated on the square V_2. To restore this condition, Chameleon proceeds with colouring the connection (V_1, V_2) *green* (Step (2), Case 2).

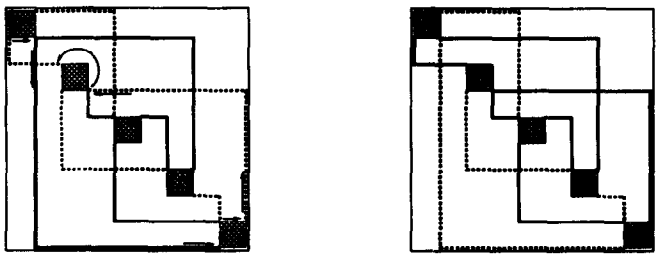

Fig. 18. A modification generated by Chameleon.

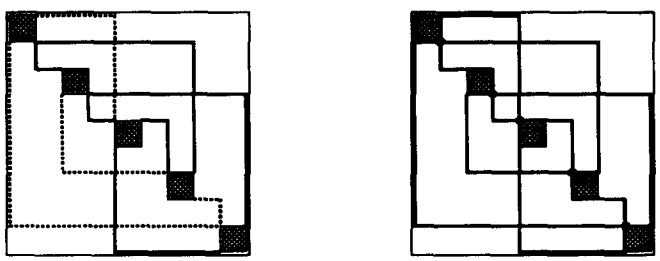

Fig. 19. A feasible colouring and an embedding.

Since this yields a violation of colouring condition (2) on V_1, the connection (V_1, V_5) will be coloured *red* instead of green.

The left picture in Fig. 18 shows the path of modification, while the picture on the right side shows the resulting colouring which fulfills the colouring conditions on each square.

After finding a feasible colouring, we are able to extend the connections along the squares' borders. Fig. 10 shows us how to do this. Thereby, it may be necessary to exchange the contacts of two connections as shown in Fig. 3 (see Fig. 14 for an example). Given a feasible colouring, we always can find a proper embedding point as well as 2-bends routings along the squares' borders. The reason for this is that the connections are balanced with respect to the inputs/outputs as well as with respect to their colours. Thus on the border of each square, *red incoming* as well as *green outgoing* connections can be routed directly while other connections may have at least one bend. Fig. 19 shows how this can be done in our example. Overall, the following algorithm Embed constructs embeddings with at most two bends per wire.

Algorithm Embed

(1) Call Balancer, but do not extend the connections along the squares' borders.
(2) Call Painter.
(3) Extend the connections along the squares' borders as shown in Fig. 10.

Since Balancer as well as Painter proceed step-by-step, crossing each square in each step at most twice, this algorithm terminates after at most $n \cdot 4n + n \cdot 4n$ steps causing an upper worst case bound for the computing time of $\mathcal{O}(n^2)$.

4. Conclusion

The presented algorithm shows that there is a large potential of improvement. Consider Fig. 10. Only the first two cases (a) and (b) cover two vertical as well as two horizontal grid edges. In case (c) one vertical and in case (d) an additional horizontal grid edge is uncovered.

This directly leads to the idea of maximizing the number of the "nice" cases (c) and (d). One possibility to achieve this is to reorder the vertices $v_1, \ldots, v_n$ since this order is kept invariant during the whole execution of the algorithm.

The other question is why to embed the vertices on the diagonal. There is a very interesting *floating horizon* approach developed by Biedl [1]. Thereby, the vertices are embedded line-by-line from the bottom to the top of the grid using a special s,t-ordering.

Acknowledgements

I would like to thank Dr. Stefan Felsner who posed this problem as an exercise in the lecture course *Discrete Mathematics* held by Professor Dr. R. Möhring. Stefan convinced me that the approach presented above is not obvious. This paper is just the consequence of his helpful questions.

References

[1] Th. Biedl, Embedding nonplanar graphs in the rectangular grid, RUTCOR at Rutgers University, New Brunswick, NJ, personal communication.

[2] S. Even and R. Tarjan, Computing an st-numbering, Theoret Comput. Sci. 2 (1976) 339–344.

[3] M. Formann and F. Wagner, The VLSI layout problem in various embedding models, Graph-Theoretic Concepts in Computer Science (16th Workshop WG'90) (Springer, Berlin, 1991) 130–139.

[4] G. Kant, Drawing graphs using the lmc-ordering, Technical Report RUU-CS-92-33, Department of Computer Science, University of Utrecht, Utrecht (1992); an extended abstract can be found in: Proceedings of the 33th Annual IEEE Symposium on Foundations of Computer Science (Pittsburgh, 1992).

[5] G. Kant, Algorithms for drawing planar graphs, Dissertation, University of Utrecht, Utrecht (1993).

[6] M.R. Kramer and J. van Leeuwen, The complexity of wire routing and finding minimum area layouts for arbitrary VLSI circuits, in: F.P. Preparata, ed., Advances in Computing Research, Vol. 2: VLSI-Theory (JAI Press, Reading, MA, 1984) 129–146.

[7] A. Lempel, S. Even and I. Cederbaum, An algorithm for planarity testing on graphs, in: P. Rosenstiehl, ed., Theory of Graphs, International Symposium, Rome 1966 (Gordon and Breach, New York, 1967) 215–232.

[8] Th. Lengauer, Combinatorical Algorithms for Integrated Circuit Layout (Teubner; Wiley, Stuttgart; New York, 1990).
[9] R. Lipton and R. Tarjan, Applications of a planar separator theorem, in: Proceedings of the 18th IEEE Symposium on FOCS (1977) 162–170.
[10] J. Storer, On minimal node cost planar embeddings, Networks 14 (1984) 181–212.
[11] L. Valiant, University considerations in VLSI circuits, IEEE Trans. Comput. 30 (1981) 135–140.